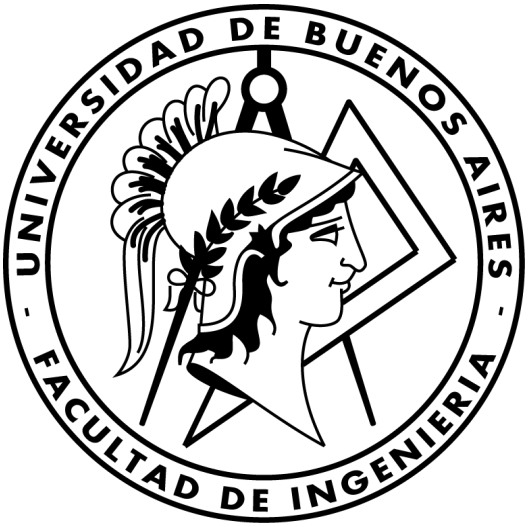
FACULTAD DE INGENIERÍA,
UNIVERSIDAD DE BUENOS AIRES
2024



SISTEMAS DISTRIBUIDOS

Informe TP1

Padrón	Alumno	Correo electrónico
108000	Juan Pablo Aschieri	jaschieri@fi.uba.ar
107870	Ugarte, Ricardo Martín	mugarte@fi.uba.ar

Cátedra: PABLO ROCA

Corrector: NICOLAS EZEQUIEL ZULAICA RIVERA

Índice

Introducción

El objetivo del siguiente documento es presentar un diseño de arquitectura para llevar a cabo un sistema distribuido. El mismo analiza reseñas de libros del sitio de Amazon y resuelve ciertas consultas necesarias para las campañas de marketing.

Para ello, se utilizarán diferentes vistas, cada una de las cuales ilustra un aspecto en particular del software desarrollado. El sistema estará optimizado para entornos multicomputadoras, así como soportará el incremento de los elementos de cómputo para escalar los volúmenes de información a procesar. No se considerará tolerancia a fallas.

Arquitectura de software

El cliente simulará ser un web scraper que va tomando libros y reseñas en tiempo real de la página de Amazon. Para ello irá leyendo un dataset de a batches y se los irá comunicando a los workers. Cada worker es responsable de realizar una operación a los datos, ya sea de filtrado, transformación o distribución. Luego envía sus resultados, ya sea a otro worker, para que continúe con más procesamiento, o al cliente para que éste lo almacene.

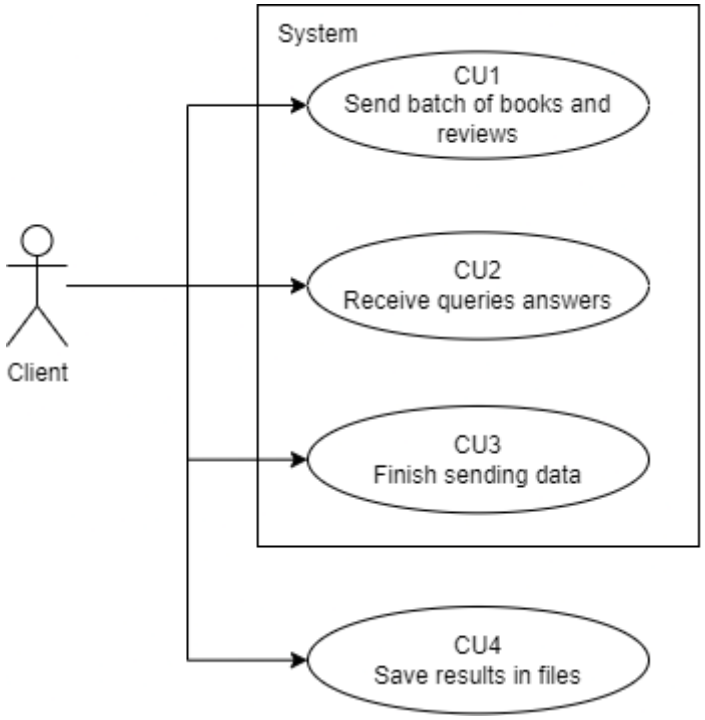
Todas las comunicaciones dentro del sistema realizadas mediante un middleware. Esto abstrae la necesidad de comunicarse mediante sockets y simplifica ambos extremos de la comunicación, permitiéndonos utilizar patrones como publisher-subscriber o producer-consumer de manera sencilla. Cabe aclarar, que el cliente y los workers no enviaran libros o reseñas de manera individual, si no que enviaran de a batches de libros y reseñas. Esto es dado que de lo contrario se estarían mandando muy pocos datos por envío, haciendo que los programas sean más ineficientes por la necesidad de pasar de kernel space a user space más veces.

Se proponen las siguientes vistas

- **Vista de escenarios:** se listan casos de uso que representen las funcionalidades centrales del sistema.
- **Vista lógica**: describe las partes significativas del modelo y qué estructuras se encuentran en él, haciendo uso de un diagrama de clases
- **Vista de procesos**: describe la descomposición del sistema en threads y procesos y cómo estos se comunican o interactúan entre sí, utilizando un diagrama de actividades.

- **Vista de desarrollo**: describe el mapeo de los paquetes en los diferentes componentes que los implementan a través de un diagrama de paquetes
- **Vista física:** describe la distribución física del sistema y cómo están relacionados sus nodos con la ayuda de un diagrama de robustez.

Vista de Escenarios



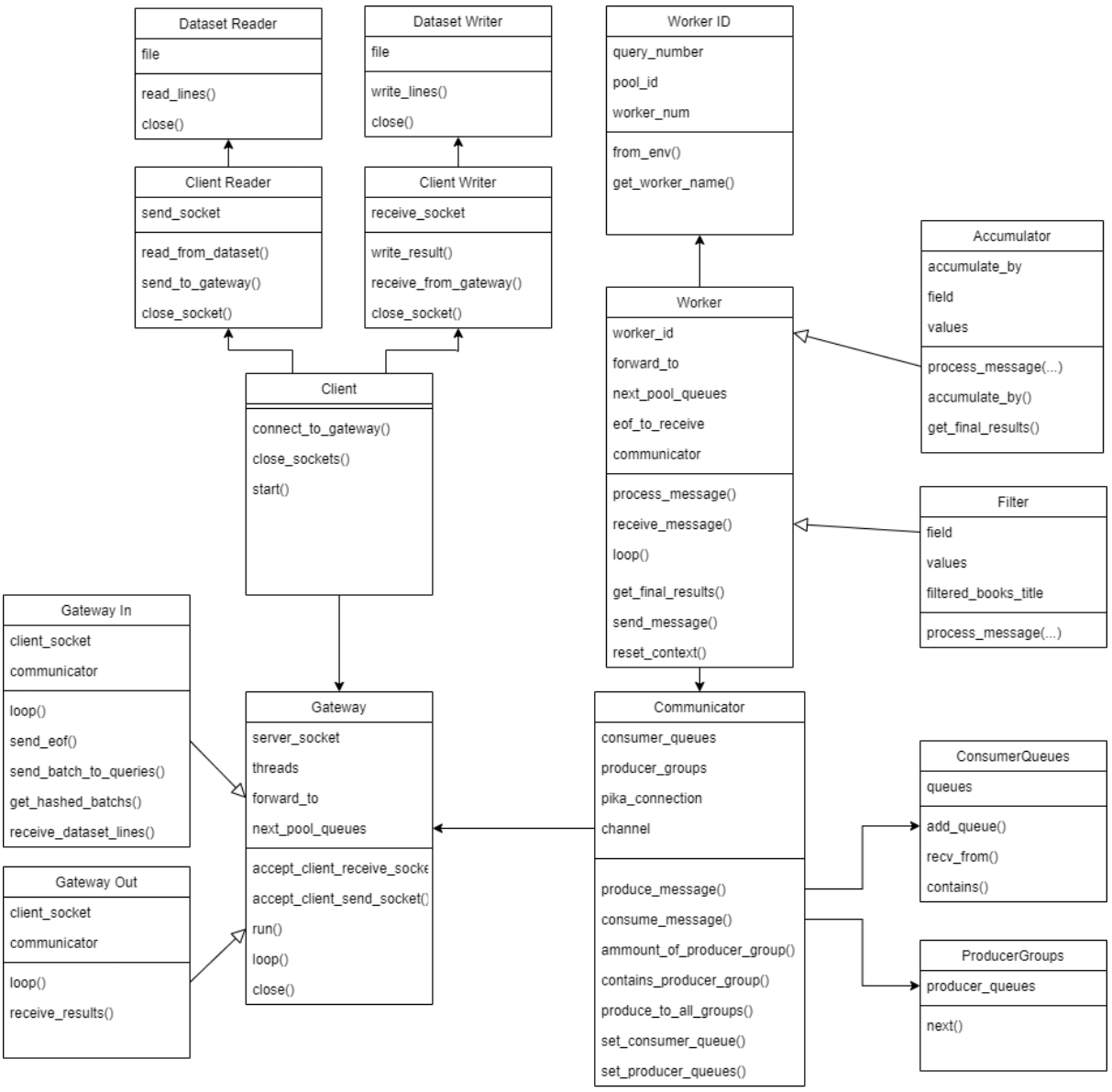
Este es un diagrama de casos de uso, identificamos al cliente como único actor capaz de realizar distintas acciones con el objetivo de resolver las siguientes consultas

1. Título, autores y editoriales de los libros de categoría “Computers” entre 2000 y 2023 que contengan ‘distributed en su título
2. Autores con títulos publicados en al menos 10 décadas distintas
3. Títulos y autores de libros publicados en los 90' con al menos 500 reseñas.
4. 10 libros con mejor rating promedio entre aquellos publicados en los 90’ con al menos 500 reseñas.
5. Títulos en categoría "Fiction" cuyo sentimiento de reseña promedio esté en el percentil 90 más alto.

Por un lado el cliente realiza actividades independientes del sistema, descargar el dataset y almacenar los resultados en su memoria interna. Por otro lado, el cliente utilizará el sistema para realizar 3 acciones:

- Enviar los datos de libros y reseñas de Amazon para el posterior procesamiento de las queries.
- Recibir los resultados que el sistema va calculando.
- Comunicar al sistema que no hay más datos a enviar, para que este calcule los resultados finales.

Vista Lógica



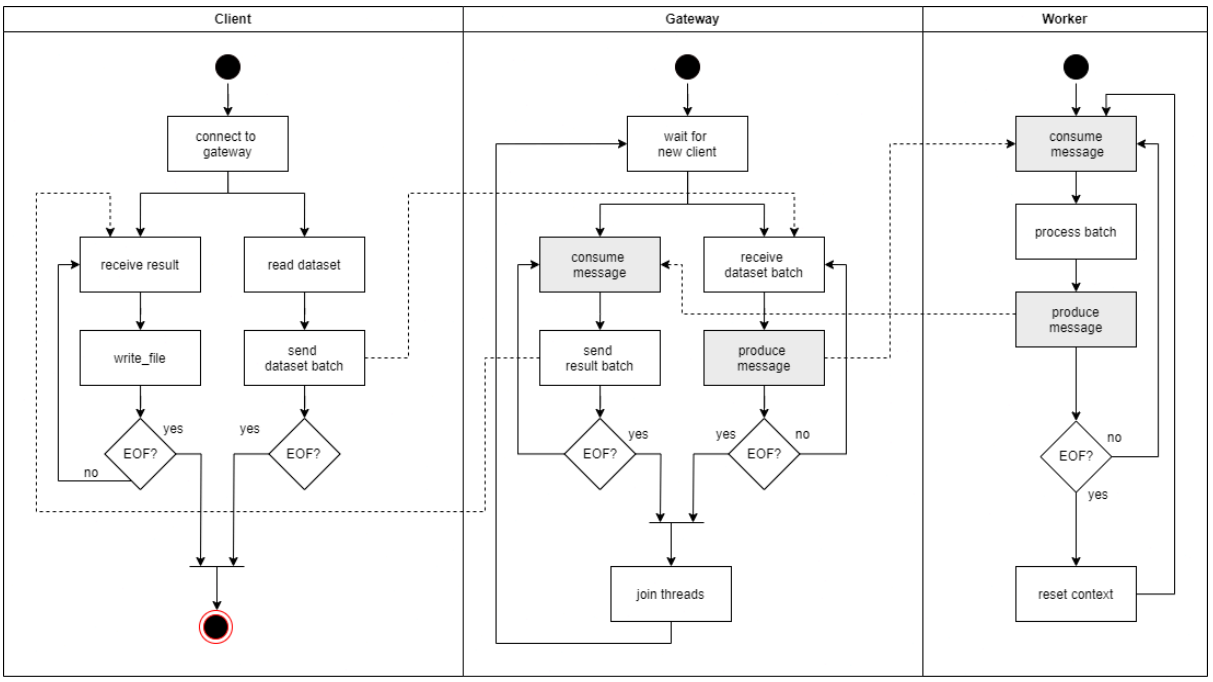
Este es el diagrama de clases y muestra las entidades que participan en el sistema. Entre las importantes se encuentran:

- **Client**: será quien actúe como el web scraper e irá transmitiendo libros y reseñas provenientes del dataset descargado del servidor de Kaggle. El cliente, usando dos

threads distintos, le irá transmitiendo estos objetos al gateway y recibiendo los resultados de las consultas mediante sockets

- **Gateway:** Solo habrá uno en el sistema, y funcionará como punto de entrada y salida del sistema. En un thread, el gateway recibirá del cliente los libros y reseñas, y luego los enviará a las cadenas de workers para su procesamiento. También es capaz mediante otro thread de recibir los resultados de las consultas para enviarselos a través de sockets al cliente.
- **Communicator:** Es el middleware para llevar a cabo la comunicación entre procesos del sistema. Utilizando las interfaces que se proveen, dos procesos que se quieran comunicar podrán hacerlo de manera mucho más simple sin tener que recurrir a manejar sockets de manera directa. El middleware permite la comunicación utilizando una versión de productor-consumidor.
    - Los productores al producir indican, además del mensaje a enviar, el id del grupo al que quieren producir. Opcionalmente se puede indicar a qué miembro puntual de ese grupo se quiere enviar. Si no se indica el worker específico, se hará round robin para determinar el próximo worker a recibir el mensaje dentro del pool. Si se indica, entonces se le enviará el mensaje a ese worker particular, lo cual es útil para cuando se quiera acumular valores y que todas las claves vayan a parar al mismo worker, ya que se pueden hashear los campos por los que se quiere acumular, y así obtener en múltiples procesos distintos los mismos workers.
    - Los consumidores para consumir simplemente indican el nombre de la cola de la cual quieren recibir mensajes y posteriormente el middleware hace un ACK del mismo.
- **Worker**: Cada uno de los workers tiene su propia cola y es responsable del procesamiento de los datos recibidos por la misma, y el posterior envío del resultado, ya sea devuelta al gateway o a otros worker que tomarán su resultado como datos. Un worker siempre sabrá, por variables de entorno, a quien o quienes tiene que forwardear el mensaje y también cuántos workers tiene la próxima pool . Una cadena de workers será capaz de resolver las queries. Hay 2 tipos de workers:
    - **Filter:** Se encarga de filtrar los datos recibidos en función de una condición en particular.
    - **Accumulator:** Recibe datos, les aplica una función y los almacena. Una vez se cumple una condición, como puede ser cantidad de datos específicos almacenados, se envían los datos. Por ejemplo, un contador sería un acumulador simple. Adicionalmente tras recibir un eof, se pueden devolver los resultados que requieren del procesamiento de todos los datos.

Vista de Procesos



Este es el diagrama de actividades que ilustra el flujo de cada proceso corriendo en el sistema.

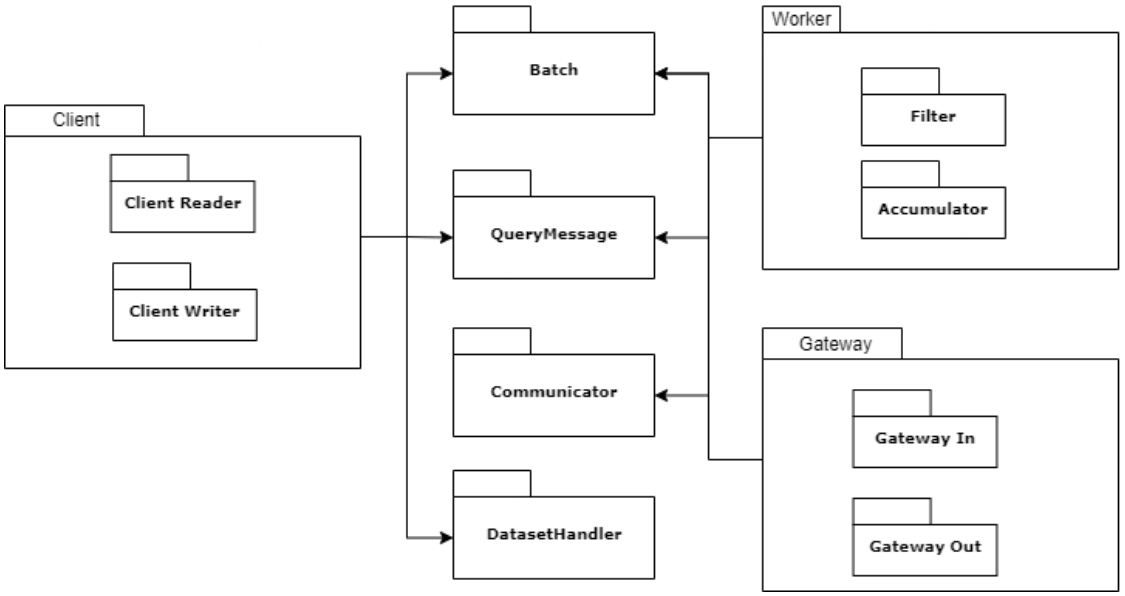
Por un lado, el cliente tendrá dos hilos de ejecución, donde uno se encarga de extraer información del dataset mientras que no se llegue al final del archivo, y luego enviar las filas de a batches de tamaño fijo al gateway. El otro thread lo que hará es quedarse escuchando nuevos resultados y según el número de query adjuntado, escribirlo en uno de los cinco archivos existentes.

El gateway como se explicó anteriormente, espera la conexión entrante de un nuevo cliente. Una vez establecido creará un threads para escuchar lo que el cliente le comunique con un socket y otro para que mediante el middleware, interactúe con los workers de la primera pool de cada query para enviarles los libros o reseñas y tras ser procesados recibir una respuesta. Cuando se detectan EOFs, se joinean los threads y se vuelve a comenzar del principio esperando un nuevo cliente.

Por último aparecen los workers donde se usa la clase madre descrita en la vista lógica para indicar un flujo genérico de los mismos. Cada vez que reciben un batch, lo procesan y

luego envían el resultado al siguiente worker mediante el middleware, o al gateway en caso de tratarse de un worker de la última pool de una query. Hay que tener en cuenta que el end of file del dataset se va propagando en los diferentes procesos hasta llegar aquí, siendo útil para terminar y que el worker reinicie sus valores y limpie su contexto para seguir escuchando para llevar a cabo futuros procedimientos. Para que un worker de por finalizado el procesamiento de un cliente, el worker debe recibir un EOF por cada worker en las pools de las que recibe mensajes. De lo contrario, si diera por finalizada la ejecución tras recibir un único EOF, podrían haber datos que no se procesen, ya que otros workers de la pool anterior podrían tener más datos a procesar.

Vista de Desarrollo

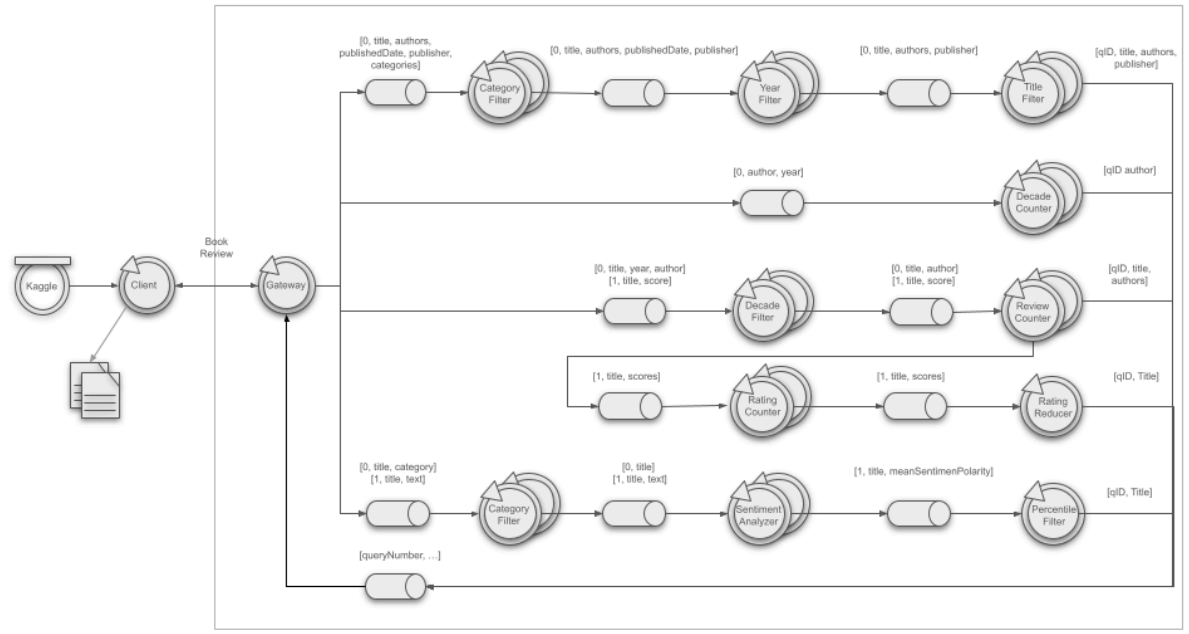


Este es el diagrama de paquetes y muestra cómo el cliente hará uso de las funcionalidades brindadas por el DatasetHandler para la lectura del dataset, así como también del paquete mensajes para llevar a cabo la serialización y deserialización de los mensajes transmitidos por socket con el gateway.

Asimismo, el worker tendrá una implementación genérica pero además traerá consigo 2 tipos de worker como se ha descrito anteriormente, pudiendo ser un filtro o un acumulador. Ambos serán capaces de distribuir por título llamando a una función de hash.

Vista Física

Diagrama de robustez



Este es el diagrama de robustez que expone el sistema distribuido entero. Todas las comunicaciones que se realizan entre los distintos componentes del sistema será realizada a través del middleware, excepto por el cliente y el gateway que usan sockets. Si se trata de un mensaje que contiene un libro, el header lo indicará con un “0”. Si es una reseña con un “1”, y si es un resultado lo indicará con el número de query correspondiente, para que al final del circuito el cliente sepa en qué archivo guardarlo

El gateway utilizará el middleware para mandar los libros y las reseñas a cada uno de los subsistemas distribuidos que se encargan de procesar las queries, donde cada worker de entrada tendrá una cola con el nombre que lo representa (cabe destacar que en el gráfico aparece una cola por cada comunicación en todo momento, pero en realidad son múltiples colas, una para cada worker).

El gateway, enviará a cada query, un mensaje QueryMessage, que estará formado por los mínimos campos necesarios que necesite cada query. En el sistema hay algunos pools de workers, que necesitan acumular todos los mensajes que tenga un valor en específico en un campo, como es el caso de los contadores de reseñas por libro. Para lograr esto se enviará a el worker cuyo id corresponda al resultado de hashear el valor.

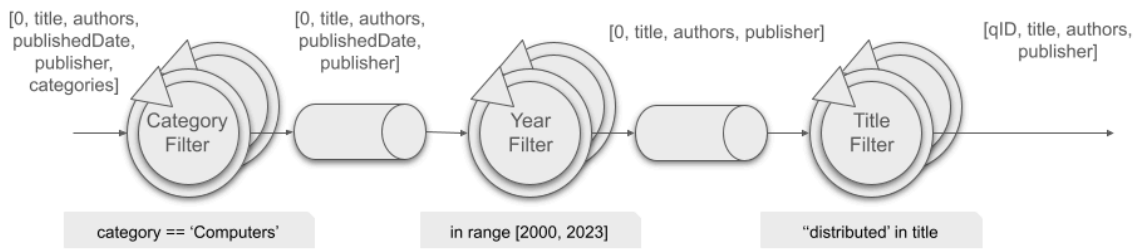
Por ejemplo, si el gateway recibe un libro con el título *“Harry Potter: La Cámara Secreta”*, iterará por cada query y aplicará la función de hash para saber a que worker de la pool correspondiente enviarselo. Para la query 1, en caso de devolver 0, querrá decir que deberá enviarle al worker “1.0.0”: el primero de la primer pool de la query 1, que está representada en el primer dígito.

Cabe destacar que a pesar de que no todos los pools necesitan esta distribución, donde tienen todos los mensajes de un campo, el sistema utiliza este para todas las comunicaciones. Esto se debe a que en un gran volumen de datos, si asumimos que el resultado de hashear algo como un título o el nombre de un autor (con SHA256), resulta en un worker aleatorio, podremos decir que se distribuyen equitativamente los datos a procesar en el próximo pool de workers.

Cada query subsystem irá procesando los libros y reseñas y de ser posible irá devolviendo resultados periódicamente por el middleware, indicando a qué query pertenece el resultado. De no poder dar resultados parciales, los subsistema de queries enviaran el resultado una vez este esté disponible, ya sea tras acumular suficientes datos o una vez el cliente envíe una señal de fin del archivo, EOF, indicando que no se enviaran más datos a procesar.

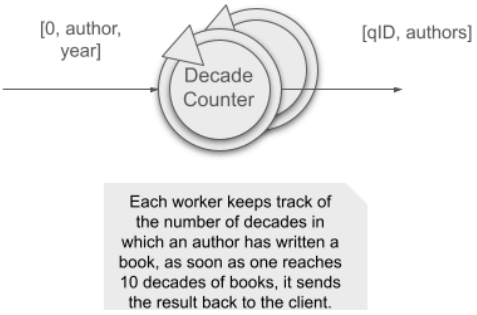
A continuación se mostraran los subsistemas de cada query

Query 1 subsystem



El sistema de la query 1, está compuesto por 3 pasos, primero un filtro por categoría, luego un filtro por año, y finalmente un filtro que chequea si ‘distributed’ está en el título del libro. Cabe destacar que todos estos filtros pueden ser paralelizados, y en vez de hablar de un único worker filtro, hablar de un pool de workers filtros. Esto es por la naturaleza de la query, ya que da lo mismo que libro sea agarrado por cualquier worker y no hay necesidad de coordinación, por lo que se puede paralelizar el trabajo sin ningún problema.

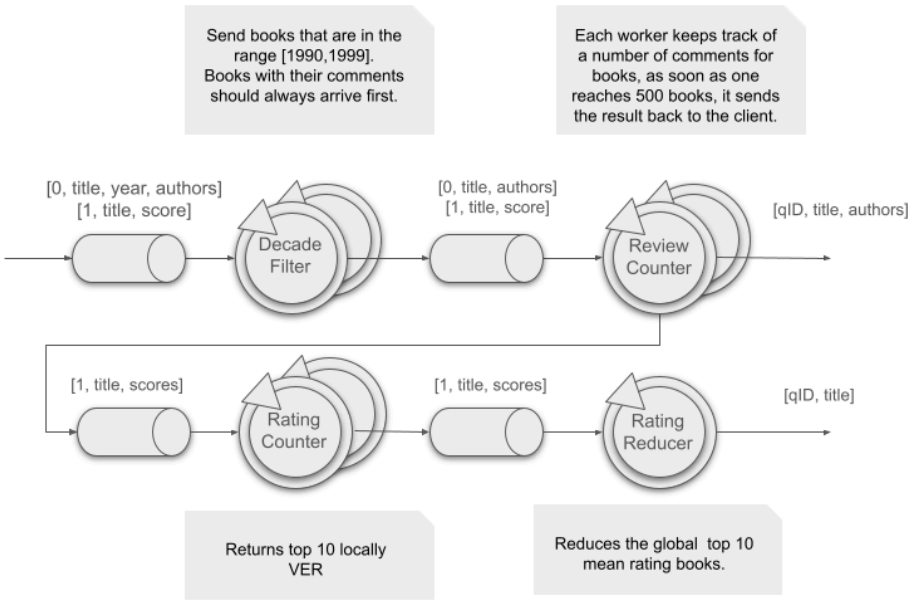
Query 2 subsystem



El sistema de queries 2 consta de contadores de décadas. Es necesario que se envíen los libros de manera distribuida por autor, es decir que a un worker, siempre le lleguen todos los libros del mismo autor. Los contadores entonces sabrán que tienen todos los libros de sus autores. Una vez reciban un libro, verificarán de qué década es, y una vez consigan 10 libros que hayan sido publicados en 10 décadas distintas enviaran el resultado al cliente.

Sin recibir los datos distribuidos por autor no sería posible paralelizar las tareas de los Decade Counters ya que no se podría garantizar que un worker tenga todos los libros de un autor, y por ende no podría verificar correctamente la condición.

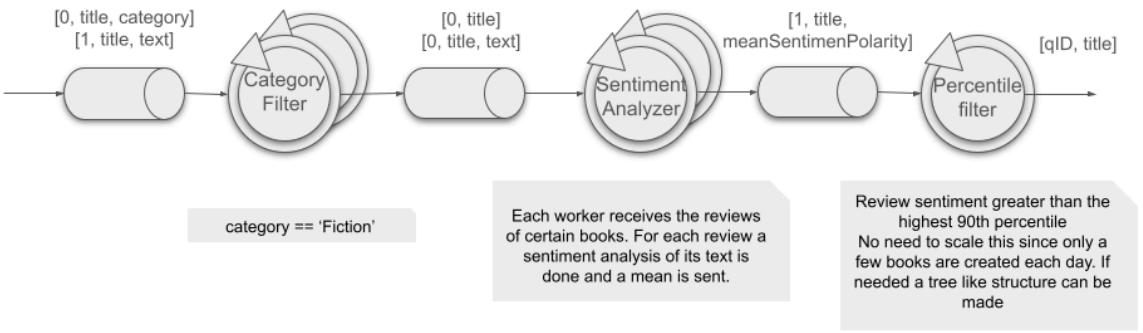
Query 3 y 4 subsystem



Similar al caso anterior, en el sistema de query 3 se deben recibir los libros distribuidos por título. La información luego pasará a un filtro de décadas para quedarse con los libros publicados en los años 90', que luego pasarán por un contador de reseñas que validará aquellos que tengan al menos 500 de ellas para mandarlas como resultado.

La query 4 al solicitar lo mismo que la query 3, solo que adicionalmente desea quedarse con el top 10 de ellos, reutiliza la query 3 y se alimenta con la salida del mismo, recibiendo el título y su respectivo atributo meanrating. Al reutilizar la salida de la query 3 nos ahorramos tener que realizar los mismos procesamientos múltiples veces.
Inicialmente la query3 solo guardaba la cantidad de reviews para un libro y una vez esta llegaba a 500, enviaba automáticamente el resultado. Para poder reutilizar la salida de la query3, para la query 4, la query 3 también almacena la suma del rating de cada review, y luego envía el promedio de los rating. Para hacer esto la query 3 deberá esperar hasta haber recibido todos los datos para poder enviar los resultados.

Query 5 subsystem



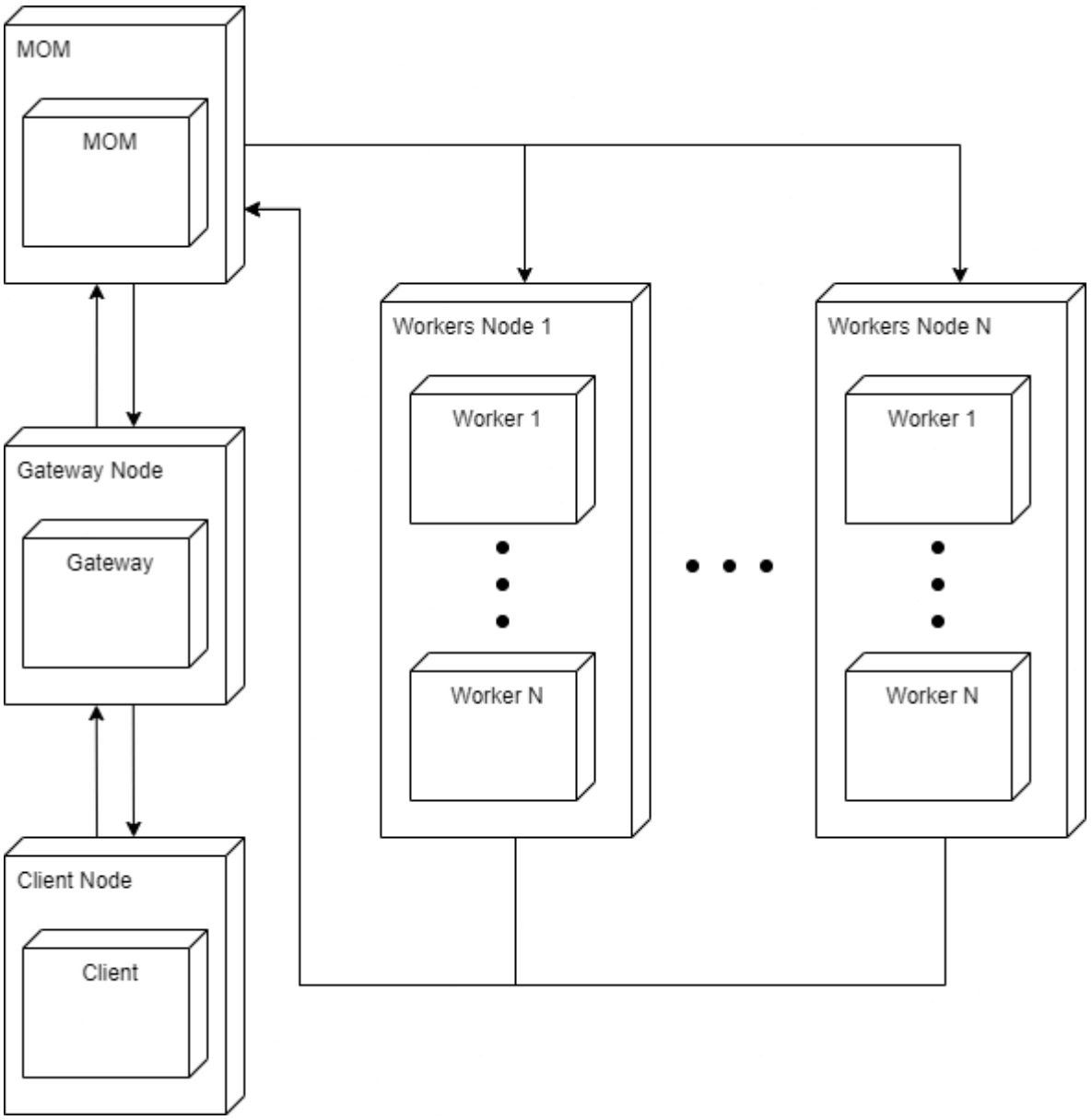
El sistema de queries 5 al igual que los anteriores, deberá recibir las queries distribuidas por título de libro. Una vez recibido los mensajes se aplica un filtro de categoría para quedarse con sólo aquellos que sean de ficción. Luego aparece un worker acumulador dedicado al análisis de sentimientos, el cual luego de realizar el análisis de las reseñas, larga como salida el título del libro y su polaridad de sentimiento promedio, en base a las reseñas asociadas al mismo. Finalmente el Percentile filter irá acumulando los promedios. Una vez recibidos todos los promedios de análisis de sentimiento de cada libro, se quedará con el 10% más alto y los enviará al cliente.

Esta query trae consigo algo que no pasaba en las anteriores queries. El último nodo es un único worker en vez de un pool de workers. Esto se debe a que si pusiéramos un pool de workers para quedarse con el 10% más alto, no conocerán los valores de los demás workers obligándolos a coordinarse entre sí para obtener el resultado final. Para solucionar la coordinación, la solución que se nos había ocurrido fue armar una estructura como árbol binario, donde dos workers se comunican entre si, juntan sus datos y mandan al próximo nivel los datos que podrían estar en el 10% más alto de todos los datos, y descartan el resto. En esta comparación de datos, si hay más de 20 workers no se podrá descartar nada, ya que la suma de los datos que ambos workers será menor igual al 10%, ya que cada uno tiene el 5% de los datos o menos (asumiendo que fueron distribuidos equitativamente). Una vez queden menos de 20 workers se empezarán a descartar valores hasta llegar a la última comparación y obtener el 10% más alto.

Sin embargo, todo este sistema complejo no nos reduce mucho la cantidad de datos que va a tener el último worker. En vez de tener un O(n) donde n es la cantidad de datos que llegan al último nodo, tendremos O(n/10), pero en múltiples workers. Por lo que si la cantidad de datos fuera escalar mucho, el sistema no escalaría con ninguna de las dos soluciones, ni la simple con un solo worker, ni la compleja con un árbol de workers.

Ahora bien, ¿hace falta escalar esta parte del sistema?. La conclusión es que no. Esto es dado a la naturaleza de los datos. Al último nodo de procesamiento de la query 5 le llegará solo un valor por cada libro de ciencia ficción: si asumimos el peor caso donde todos los libros del dataset son de ciencia ficción, tendremos 300 MB de data a procesar, lo que no es mucho, por lo que un único worker lo podría procesar. Aún más, si asumimos que el sistema está andando en la vida real y recibe libros y reseñas a medida que estos son publicados, el último nodo no recibirá más de algunos libros por día. En definitiva, el sistema debe escalar principalmente para aquellos procedimientos que respecta a las queries que manejan reseñas ya que su volumen de datos es órdenes de magnitud más grande que los libros. Es por esto que podemos ahorrarnos la complejidad extra de armar un árbol de workers que se comuniquen entre sí para obtener el percentil 90 y simplemente procesar todo en uno.

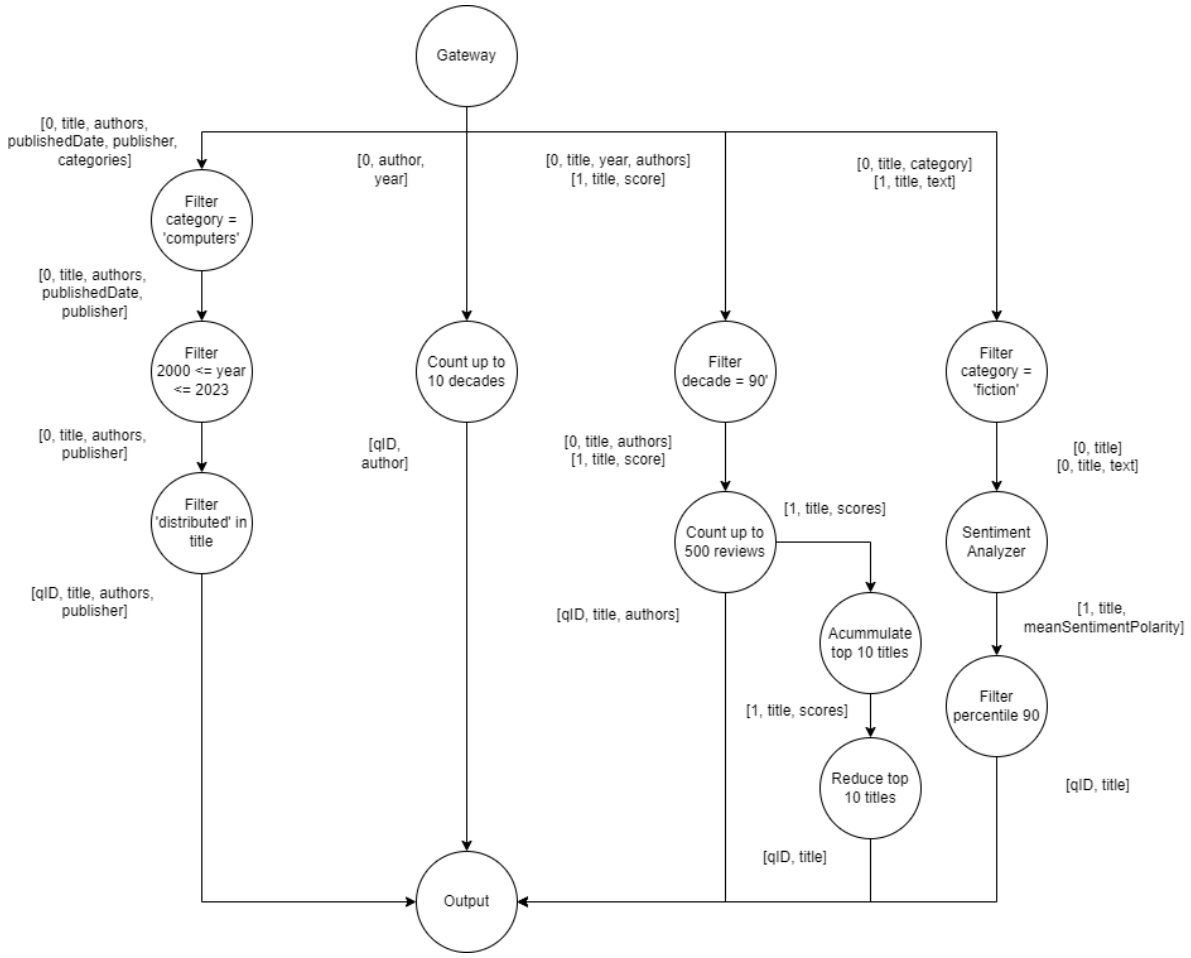
Diagrama de despliegue



Este es un diagrama de despliegue, nos muestra cómo se distribuyen los procesos del sistema en distintos componentes físicos. En este caso, tendremos 4 tipos de nodos de cómputo. Por un lado un único nodo donde se ejecutará el código del cliente, quien será quien envíe activamente los datos del sitio web. Luego tendremos un nodo que tenga al gateway, que será el que funcione como entrada al sistema y recibirá las peticiones del cliente. Como núcleo central de las comunicaciones del sistema habrá un MOM por el cual pasarán todas las comunicaciones (nosotros utilizaremos RabbitMQ). Finalmente, debido a la escalabilidad del sistema, se podrán agregar tantos nodos de workers cómo uno quiera. En cada uno de ellos, podrá ejecutar tambien n instancias de los mismos, es decir que en un mismo nodo de cómputo podríamos tener múltiples filtros y acumuladores. Cabe

destacar, que hay acumuladores que realizan análisis de sentimiento, un procesamiento pesado. Por lo que se querría que no hubiera más de un worker de análisis de sentimiento por, si no que estos estén distribuidos en múltiples nodos para aprovechar mejor el uso de CPU.

Gráfico Acíclico Dirigido



Por último, presentamos el DAG, donde se ve a partir del gateway el recorrido que realizan los datos a medida que pasan por los diferentes workers. Cada nodo dentro del gráfico, representa una operación a realizar sobre los datos, llevada a cabo por un pool de workers. De esta forma, vemos que hay 5 salidas, una por query, sin embargo hay solo 4 entradas. Esto se debe a que algunas queries deben realizar el mismo procesamiento a los datos, por lo que para evitar repeticiones, decidimos juntar este flujo en un único pool de workers.

Cada nodo únicamente enviará los datos que necesite el próximo proceso, y descartará aquellos que no sean necesarios. Si bien todos los datos en el gráfico llegan al output, estos

no llegarán al mismo tiempo, ya que algunas queries podrán dar el resultado inmediatamente, mientras que otras deberán acumular una cantidad de datos antes de poder continuar con el procesamiento y dar la respuesta.

Por último, trabajamos bajo la suposición de que siempre llegarán los mensajes de libros antes que sus reseñas. Esto es para evitar guardar temporalmente reseñas de libros que aún no han llegado para luego asignarlos.

Distribución de tareas

Tarea	**Encargado**
Clase DatabaseHandler	Pablo
Clase Worker	Ambos
Subclase Filtro	Pablo
Subclase Acumulador - Contadores	Martin
Subclase Acumulador - Análisis de sentimientos	Martin
Biblioteca Interfaz de comunicación del middleware	Ambos
Subsistema Query 1	Martin
Subsistema Query 2	Pablo
Subsistema Query 3 y 4	Pablo
Subsistema Query 5	Martin
Elaboración del cliente	Pablo
Elaboración del gateway	Martin
Unión de partes	Ambos
Testing (durante todo el proceso)	Ambos